An Interval Extension Based on Occurrence Grouping: Method and Properties

Ignacio Araya, Bertrand Neveu, Gilles Trombettoni

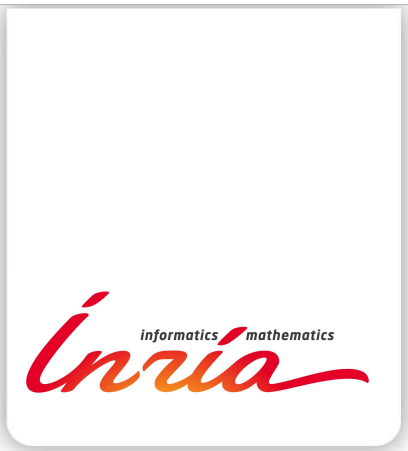
RESEARCH REPORT

N° 7806

November 2011

Project-Team COPRIN

ISSN 0249-6399 ISRN INRIA/RR--7806--FR+ENG



An Interval Extension Based on Occurrence Grouping: Method and Properties

Ignacio Araya*, Bertrand Neveu†, Gilles Trombettoni‡

Project-Team COPRIN

Research Report n° 7806 — November 2011 — 26 pages

Abstract: In interval arithmetics, special care has been brought to the definition of interval extension functions that compute narrow interval images. In particular, when a function f is monotonic w.r.t. a variable in a given domain, it is well-known that the monotonicity-based interval extension of f computes a sharper (interval) image than the natural interval extension does.
This paper presents a so-called "occurrence grouping" interval extension $[f]_{og}$ of a function f. When f is *not* monotonic w.r.t. a variable x in a given domain, we try to transform f into a new function f^{og} that is monotonic w.r.t. two subsets x_a and x_b of the occurrences of x: f^{og} is increasing w.r.t. x_a and decreasing w.r.t. x_b. $[f]_{og}$ is the interval extension by monotonicity of f^{og} and produces a sharper interval image than the natural extension does.
For finding a good occurrence grouping, we propose a linear program and an algorithm that minimize a Taylor-based over-estimate of the image diameter of $[f]_{og}$. Experiments show the benefits of this new interval extension for solving systems of nonlinear equations.

Key-words: intervals, interval extension, monotonicity, occurrence grouping

---

This research report is the extended version of a paper appearing in the Computing journal (Springer).

* UTFSM, Chile. Email: iaraya@inf.utfsm.cl
† Imagine LIGM Université Paris–Est, France. Email: Bertrand.Neveu@enpc.fr
‡ INRIA, I3S, Université Nice–Sophia, France. Email: Gilles.Trombettoni@inria.fr

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Une extension aux intervalles basée sur le regroupement d'occurrences : méthode et propriétés

Résumé : L'analyse d'intervalles a proposé plusieurs extensions aux intervalles qui essaient de calculer des images étroites des fonctions. En particulier, quand une fonction f est monotone par rapport à une variable sur un domaine donné, il est bien connu que l'*extension aux intervalles par monotonie* de f permet de calculer un intervalle image plus étroit que l'*extension naturelle*.

Cet article présente une nouvelle extension aux intervalles d'une fonction f appelée *regroupement d'occurrences* et notée $[f]_{og}$. Quand f n'est pas monotone par rapport à une variable x sur un domaine donné, nous essayons de transformer f en une nouvelle fonction f^{og} qui est monotone par rapport à deux sous-ensembles x_a et x_b des occurrences de x : f^{og} est croissante par rapport à x_a et décroissante par rapport à x_b. $[f]_{og}$ est l'extension aux intervalles par monotonie de f^{og} et produit une image plus étroite que l'extension naturelle.

Pour trouver un bon regroupement d'occurrences, nous proposons un programme linéaire et un algorithme qui minimisent une surestimation du diamètre de l'image de $[f]_{og}$ basée sur une forme de Taylor de f. Finalement, des expérimentations montrent les avantages de cette nouvelle extension pour la résolution de systèmes d'équations non linéaires.

Mots-clés : intervalles, extension aux intervalles, monotonie, regroupement d'occurrences

Contents

1 Introduction

The computation of sharp interval image enclosures is in the heart of interval arithmetics [15]. It allows a computer to evaluate a mathematical formula while taking into account in a reliable way round-off errors due to floating point arithmetics. Sharp enclosures also allow combinatorial interval methods to quickly converge towards the solutions of a system of constraints over the reals. At every node of the search tree, a *test of existence* checks that, for every equation $f(X) = 0$, the interval extension of f returns an interval including 0 (otherwise the branch is cut). Also, *constraint propagation* algorithms, used at every node of the search tree to reduce the search space, can be improved when they use better interval extensions. For instance, the `Box` constraint propagation algorithm [3] uses a *test of existence* inside its iterative splitting process.

This paper proposes a new interval extension and we first recall basic material about interval arithmetics [15, 16, 10] to introduce the interval extensions useful in our work.

An interval $[x] = [a,b]$ is the set of real numbers between a and b. $\underline{x} = a$ denotes the minimum of $[x]$ and $\overline{x} = b$ denotes the maximum of $[x]$. The diameter/width of an interval is: $diam([x]) = \overline{x} - \underline{x}$, and the absolute value of an interval is: $|[x]| = max(|\overline{x}|, |\underline{x}|)$. A Cartesian product of intervals is named a *box*, and is denoted by a vector. $[V] = \{[x_1], [x_2], ..., [x_n]\}$. (Vectorial variables appear in upper case in this article.)

An *interval function* $[f]$ is a function from $\mathbb{IR}$ to $\mathbb{IR}^n$, $\mathbb{IR}$ being the set of all the intervals over $\mathbb{R}$. When a function f is a composition of elementary functions, an *extension* of f to intervals must be defined to ensure a conservative image computation.

Definition 1 (Extension of a function to $\mathbb{IR}$; also called *inclusion function*)
Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
$[f] : \mathbb{IR}^n \rightarrow \mathbb{IR}$ *is an* **extension** *of* f *to intervals iff:*

$$\begin{array}{ll} \forall [V] \in \mathbb{IR}^n & [f]([V]) \supseteq \Im f([V]) \equiv \{f(V),\, V \in [V]\} \\ \forall V \in \mathbb{R}^n & f(V) = [f](V) \end{array}$$

The first idea is to use *interval arithmetics*. Interval arithmetics extends to intervals arithmetic operators $+$, $-$, $\times$, $/$ and elementary functions (*power*, *exp*, *log*, *sin*, *cos*, ...). For instance, $[a,b]+[c,d] = [a+c, b+d]$. The *natural interval extension* $[f]_n$ of a real function f simply replaces arithmetic over the reals by interval arithmetic.

The *optimal* image $[f]_{opt}([V])$ is the sharpest interval containing $\Im f([V])$. If f is continuous inside a box $[V]$, the natural evaluation of f (i.e., the computation of $[f]_n([V])$) yields the optimal image when each variable occurs only once in f. When a variable appears several times in f, the evaluation by interval arithmetics generally produces an over-estimate of $[f]_{opt}([V])$, because the correlation between the occurrences of a same variable is lost. Two occurrences of a variable are handled as independent variables. For example $[x]-[x]$, with $[x] = [0,1]$ gives the result $[-1,1]$, instead of $[0,0]$, as does $[x]-[y]$, with $[x] = [0,1]$ and $[y] = [0,1]$. Thus, multiple occurrences of variables render NP-hard the computation of the *optimal* image of a polynomial [11]. This main drawback of interval arithmetics causes a real difficulty for implementing efficient interval-based solvers. There exist many possible interval extensions for a function, the difficulty being to define an extension that computes a sharp approximation of the optimal image at a low cost.

The interval first-order *Taylor extension* $[f]_t$ of f, also called centered form [15], uses a Taylor form of f:

$$[f]_t([V]) = f(V_m) + \sum_{i=1}^{n} \left(\left[\frac{\partial f}{\partial x_i} \right] ([V]) \;.\; ([x_i] - x_i^m) \right)$$

where n is the number of variables in f, x_i^m is the value in the middle of the interval $[x_i]$, V_m is the n-dimensional point in the middle of $[V]$ (i.e., $V_m = (x_1^m, ..., x_k^m)$) and $\left[\frac{\partial f}{\partial x_i} \right]$ is an interval extension of $\frac{\partial f}{\partial x_i}$. The Taylor extension generally calculates sharp evaluations when the diameters of the partial derivatives are close to 0. In other cases it can be even worse than the natural extension.

A well-known variant of the Taylor extension, called here *Hansen extension* $[f]_h$, computes a sharper image at a higher cost [8]: $[f]_h([V]) \subseteq [f]_t([V])$.

Another extension to intervals uses the monotonicity of a function in a given domain. When f is monotonic w.r.t. a subset of variables, one can replace, in the natural evaluations, the intervals of these monotonic variables[1] by degenerated intervals reduced to their maximal or minimal values [15, 8].

Definition 2 (f_{min}, f_{max}, monotonicity-based extension)
Let f be a function defined on variables V of domains $[V]$. Let $X \subseteq V$ be a subset of monotonic variables. Consider the values x_i^+ and x_i^- such that: if $x_i \in X$ is an increasing (resp. decreasing) variable, then $x_i^- = \underline{x_i}$ and $x_i^+ = \overline{x_i}$ (resp. $x_i^- = \overline{x_i}$ and $x_i^+ = \underline{x_i}$).

[1] For the sake of conciseness, we sometimes write that a "variable x is monotonic" instead of writing that f is monotonic w.r.t. x.

Consider $W = V \setminus X$ the set of variables not detected monotonic. Then, f_{min} and f_{max} are functions defined by:

$$\begin{aligned} f_{min}(W) &= f(x_1^-,...,x_n^-,W) \\ f_{max}(W) &= f(x_1^+,...,x_n^+,W) \end{aligned}$$

Finally, the monotonicity-based extension $[f]_m$ of f in the box $[V]$ produces the following interval image:

$$[f]_m([V]) = \left[\underline{[f_{min}]_n([W])}, \overline{[f_{max}]_n([W])}\right]$$

The image $[f]_m([V])$ obtained by the monotonicity-based extension is sharper than, or equal to, the image $[f]_n([V])$ obtained by natural evaluation. That is:

$$[f]_{opt}([V]) \subseteq [f]_m([V]) \subseteq [f]_n([V])$$

In addition, when a function is monotonic w.r.t. each of its variables, i.e., when W is empty in Def. 2, the problem of multiple occurrences disappears and the evaluation (using a monotonicity-based extension) becomes optimal: $[f]_m([V]) = [f]_{opt}([V])$.

Note that the bounds of the evaluation by monotonicity can be computed using any interval extension, not necessarily the natural extension. When the bounds are computed with the Taylor (resp. Hansen) extension, we denote this variant by $[f]_{m+t}$ (resp. $[f]_{m+h}$).

When the evaluation by monotonicity uses, recursively, the same evaluation by monotonicity for computing the bounds of the image, we call it *recursive evaluation by monotonicity* and denote it by $[f]_{mr}$.

Consider for instance the function $f(x_1,x_2) = -6x_1 + x_1x_2^2 + 3x_2$. The partial derivatives w.r.t. x_1 and x_2 are resp. $\frac{\partial f}{\partial x_1}(x_2) = -6 + x_2^2$ and $\frac{\partial f}{\partial x_2}(x_1,x_2) = 2x_1x_2 + 3$. If the intervals of the variables are $[x_1] = [-2,-1]$ and $[x_2] = [0,1]$, then f is decreasing w.r.t. x_1 and not monotonic w.r.t. x_2. Thus, the recursive evaluation by monotonicity will compute:

$$[f]_{mr}([V]) = [\underline{[f]_{mr}(-1,[0,1])}, \overline{[f]_{mr}(-2,[0,1])}]$$

In the box $\{-1\} \times [0,1]$, related to the left bound of the evaluation, it turns out that f is now increasing w.r.t. x_2 ($\frac{\partial f}{\partial x_2}(-1,[0,1]) = [1,3]$). One can thus replace $[x_2]$ by its lower bound. On the other hand, in the box $\{-2\} \times [0,1]$, f is still not monotonic w.r.t. x_2. Overall, the recursive evaluation by monotonicity finally computes:

$$[f]_{mr}([V]) = [\underline{[f](-1,0)}, \overline{[f](-2,[0,1])}] = [6,15]$$

where $[f]$ can be any interval extension, e.g., the natural extension.

The recursive evaluation by monotonicity computes images sharper than or equal to the evaluation by monotonicity does (i.e., $[f]_{mr}([V]) \subseteq [f]_m([V])$) at a cost between 2 and $2n$ higher. In this example, we can check than the evaluation by monotonicity provides a slightly worse evaluation ($[f]_m([V]) = [5,15]$) than the recursive evaluation by monotonicity.

Contribution

This paper explains how to use monotonicity when a function is *not* monotonic w.r.t. a variable x, but is monotonic w.r.t. a *subgroup of occurrences* of x. We present in the

next section the idea of grouping the occurrences into three sets, increasing, decreasing and non monotonic auxiliary variables. Linear programs for obtaining "interesting" occurrence groupings are described in Sections 3 and 4. In Section 5, we propose an algorithm to solve the linear programming problem presented in Section 4. Finally, in Section 7, experiments show that this new occurrence grouping interval extension function compares favorably with existing ones and show its benefits for solving systems of equations, in particular when we use a filtering algorithm like `Mohc` [2, 6] exploiting monotonicity.

2 Evaluation by monotonicity with occurrence grouping

In this section, we study the case of a function which is not monotonic w.r.t. a variable with multiple occurrences. We can, without loss of generality, limit the study to a function of one variable: the generalization to a function of several variables is straightforward, the evaluations by monotonicity being independent.

Example 1 *Consider* $f_1(x) = -x^3 + 2x^2 + 14x$. *We want to calculate a sharp evaluation of this function when x falls in* $[-2,1]$. *The derivative of* f_1 *is* $f_1'(x) = -3x^2 + 4x + 14$ *and contains a positive term* (14), *a negative term* $(-3x^2)$ *and the term* $4x$ *that is negative when* $x \in [-2,0]$ *and positive when* $x \in [0,1]$. $[f_1]_{opt}([V])$ *is* $[-13.18, 15]$, *but we cannot obtain it directly by a simple interval function evaluation (one needs to solve* $f_1'(x) = 0$, *which is difficult in the general case). In the interval* $[-2,1]$, f_1 *is not monotonic. The natural interval evaluation yields* $[-29,30]$, *the Horner evaluation* $[-34,17]$ *(see [9]).*

When a function is not monotonic w.r.t. a variable x, it sometimes appears that it is monotonic w.r.t. some occurrences. A first *naive* idea leads to replace the function f by a function f^{nog}, grouping all increasing occurrences into one variable x_a, all decreasing occurrences into one variable x_b, and the non monotonic occurrences into x_c. The domain of the new auxiliary variables is the same: $[x_a] = [x_b] = [x_c] = [x]$. However, the evaluation by monotonicity of the new function f^{nog} always provides the same result as the natural evaluation.

The main idea is then to change this grouping in order to reduce the dependency problem and obtain sharper evaluations.

We can indeed group some occurrences (increasing, decreasing, or non monotonic) into an increasing variable x_a *(resp. a decreasing variable* x_b*) as long as the function remains increasing (resp. decreasing) w.r.t. this variable* x_a *(resp.* x_b*). If one can move a non monotonic occurrence into a monotonic group, the evaluation will be better (or remain the same).* Also, if it is possible to transfer all decreasing occurrences into the increasing part, the dependency problem will now occur only on the occurrences in the increasing and non monotonic parts.

For f_1, if we group together the positive derivative term with the derivative term containing zero, we obtain the new function: $f_1^{og}(x_a, x_b) = -x_b^3 + 2x_a^2 + 14x_a$. As the interval derivative of the grouping of the first two occurrences (the variable x_a) is positive: $4[x] + 14 = [6,18]$, f_1^{og} is increasing w.r.t. x_a. We can then achieve the evaluation by monotonicity and obtain the interval $[-21,24]$. We can in the same manner obtain $f_1^{og}(x_a, x_c) = -x_a^3 + 2x_c^2 + 14x_a$, the evaluation by monotonicity yields then $[-20,21]$. We remark that we find sharper images than the natural evaluation of f_1 does. In Section 3, we present a linear program to perform *occurrence grouping* automatically.

Interval extension by occurrence grouping

Consider the function $f(x)$ with multiple occurrences of x. We obtain a function $f^{og}(x_a,x_b,x_c)$ by replacing in f every occurrence of x by one of the three variables x_a, x_b, x_c, such that f^{og} is increasing w.r.t. x_a in $[x]$, and f^{og} is decreasing w.r.t. x_b in $[x]$. Then, we define the *interval extension by occurrence grouping* of f by:

$$[f]_{og}([V]) := [f^{og}]_m([V]).$$

Unlike the natural interval extension and the interval extension by monotonicity, the interval extension by occurrence grouping is not unique for a function f since it depends on the occurrence grouping (og) that transforms f into f^{og}.

3 A 0,1 linear program achieving occurrence grouping

In this section, we propose a method for automatizing occurrence grouping. Using the Taylor extension, we first compute an over-estimate of the diameter of the image computed by $[f]_{og}$. Then, we propose a linear program performing a grouping that minimizes this over-estimate.

3.1 Taylor-based over-estimate

First, as f^{og} is not (detected) monotonic w.r.t. x_c, the evaluation by monotonicity considers the occurrences of x_c as different variables such as the natural evaluation would do.

Proposition 1 *([15]) Let $f(x)$ be a continuous function in a box $[V]$ with a set of occurrences of x: $\{x_1,x_2,...,x_k\}$. $f^\circ(x_1,..,x_k)$ is a function obtained from f by considering all the occurrences of x as different variables. Then, $[f]_n([V]) = [f^\circ]_{opt}([V])$.*

Second, as f^{og} is monotonic w.r.t. x_a and x_b, the evaluation by monotonicity of these variables is optimal.

Proposition 2 *([15]) Let $f(x_1,...,x_n)$ be a monotonic function w.r.t. each of its variables in a box $[V] = \{[x_1],...,[x_n]\}$. Then, the evaluation by monotonicity computes $[f]_{opt}([V])$.*

Using these propositions, we observe that $[f^{og}]_m([x_a],[x_b],[x_c])$ is equivalent to $[f^\circ]_{opt}([x_a],[x_b],[x_{c_1}],...,[x_{c_{ck}}])$, considering each occurrence of x_c in f^{og} as an independent variable x_{c_j} in f°, ck being the number of occurrences of x_c in f^{og}. Using the Taylor evaluation, an upper bound of $diam([f]_{opt}([V]))$ is given by the right side of (1) in Proposition 3.

Proposition 3 *Let $f(x_1,...,x_n)$ be a function with domains $[V] = \{[x_1],...,[x_n]\}$. Then,*

$$diam([f]_{opt}([V])) \leq \sum_{i=1}^{n} \big(diam([x_i]) \cdot |[g_i]([V])|\big) \tag{1}$$

where $[g_i]$ is an interval extension of $g_i = \frac{\partial f}{\partial x_i}$.

Using Proposition 3, we can calculate an upper bound of the *diameter* of $[f]_{og}([V]) = [f^{og}]_m([V]) = [f^\circ]_{opt}([V])$:

$$diam([f]_{og}([V])) \leq diam([x])\Big(|[g_a]([V])| + |[g_b]([V])| + \sum_{i=1}^{ck} |[g_{c_i}]([V])|\Big)$$

where $[g_a]$, $[g_b]$ and $[g_{c_i}]$ are the interval extensions of $g_a = \frac{\partial f^{og}}{\partial x_a}$, $g_b = \frac{\partial f^{og}}{\partial x_b}$ and $g_{c_i} = \frac{\partial f^{og}}{\partial x_{c_i}}$ respectively. $diam([x])$ is factorized because $[x] = [x_a] = [x_b] = [x_{c_1}] = ... = [x_{c_{ck}}]$.

In order to respect the monotonicity conditions required by f^{og}: $\frac{\partial f^{og}}{\partial x_a}([V]) \geq 0$, $\frac{\partial f^{og}}{\partial x_b}([V]) \leq 0$, we have the sufficient conditions $\underline{[g_a]([V])} \geq 0$ and $\overline{[g_b]([V])} \leq 0$, implying $|[g_a]([V])| = \overline{[g_a]([V])}$ and $|[g_b]([V])| = -\underline{[g_b]([V])}$. Finally:

$$diam([f]_{og}([V])) \leq diam([x])\Big(\overline{[g_a]([V])} - \underline{[g_b]([V])} + \sum_{i=1}^{ck} |[g_{c_i}]([V])|\Big) \tag{2}$$

3.2 A linear program

We want to transform f into a new function f^{og} that minimizes the right side of the relation (2). The problem can be easily transformed into the following integer linear program:

Find the values r_{a_i}, r_{b_i} and r_{c_i} for each occurrence x_i that minimize

$$G = \overline{[g_a]([V])} - \underline{[g_b]([V])} + \sum_{i=1}^{k} \big(|[g_i]([V])| r_{c_i}\big) \tag{3}$$

subject to:

$$\underline{[g_a]([V])} \geq 0 \tag{4}$$

$$\overline{[g_b]([V])} \leq 0 \tag{5}$$

$$r_{a_i}, r_{b_i}, r_{c_i} \in \{0,1\};\ \ r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1,...,k, \tag{6}$$

where a value r_{a_i}, r_{b_i} or r_{c_i} equal to 1 indicates that the occurrence x_i in f will be replaced, respectively, by x_a, x_b or x_c in f^{og}. k is the number of occurrences of x, $[g_a]([V]) = \sum_{i=1}^{k} [g_i]([V]) r_{a_i}$, $[g_b]([V]) = \sum_{i=1}^{k} [g_i]([V]) r_{b_i}$, and $[g_1]([V]), ..., [g_k]([V])$ are the derivatives w.r.t. each occurrence.

We can remark that $[g_a]([V])$ and $[g_b]([V])$ are calculated using only the derivatives of f w.r.t. each occurrence of x (i.e., $[g_i]([V])$).

Linear program corresponding to Example 1

We have $f_1(x) = -x^3 + 2x^2 + 14x$, $f_1'(x) = -3x^2 + 4x + 14$ with $x \in [-2,1]$. The gradient is: $[g_1]([-2,1]) = [-12,0]$, $[g_2]([-2,1]) = [-8,4]$ and $[g_3]([-2,1]) = [14,14]$. Then, the linear program is:

Find the values r_{a_i}, r_{b_i} and r_{c_i} that minimize

$$\begin{aligned} G &= \sum_{i=1}^{3} \overline{[g_i]([V])} r_{a_i} - \sum_{i=1}^{3} \underline{[g_i]([V])} r_{b_i} + \sum_{i=1}^{3} \left(|[g_i]([V])| r_{c_i} \right) \\ &= (4r_{a_2} + 14r_{a_3}) + (12r_{b_1} + 8r_{b_2} - 14r_{b_3}) + (12r_{c_1} + 8r_{c_2} + 14r_{c_3}) \end{aligned}$$

subject to:

$$\sum_{i=1}^{3} \underline{[g_i]([V])} r_{a_i} = -12r_{a_1} - 8r_{a_2} + 14r_{a_3} \geq 0; \ \sum_{i=1}^{3} \overline{[g_i]([V])} r_{b_i} = 4r_{b_2} + 14r_{b_3} \leq 0$$

$$r_{a_i}, r_{b_i}, r_{c_i} \in \{0,1\}; \quad r_{a_i} + r_{b_i} + r_{c_i} = 1 \quad \text{for } i = 1, ..., 3$$

We obtain the minimum 22, and the solution $r_{a_1} = 1$, $r_{b_1} = 0$, $r_{c_1} = 0$, $r_{a_2} = 0$, $r_{b_2} = 0$, $r_{c_2} = 1$, $r_{a_3} = 1$, $r_{b_3} = 0$, $r_{c_3} = 0$, which is the last solution presented in Section 2. Note that the value of the over-estimate of $diam([f]_{og}([V]))$ is equal to 66 ($22 * diam[-2,1]$) whereas $diam([f]_{og}([V])) = 41$. Although the over-estimate is rough, the heuristic works rather well on this example. Indeed, $diam([f]_{n}([V])) = 59$ and $diam([f]_{opt}([V])) = 28.18$.

4 A linear programming problem achieving a better occurrence grouping

The linear program above is a 0,1 linear program and is known to be NP-hard in general. We can render it tractable while, more important in practice, improve the minimum G by allowing r_{a_i}, r_{b_i} and r_{c_i} to get *real* values. In other words, we allow each occurrence of x in f to be replaced by a *convex linear combination* of auxiliary variables, x_a, x_b and x_c such that f^{og} is increasing w.r.t. x_a and decreasing w.r.t. x_b in $[x]$. (f^{og} is not (detected) monotonic w.r.t. x_c.)

Definition 3 *(**Interval extension by occurrence grouping**)*
Let $f(x)$ be a function with multiple occurrences of the variable x. $f^{og}(x_a, x_b, x_c)$ is the function obtained by replacing in f every occurrence of x by $r_{a_i}x_a + r_{b_i}x_b + r_{c_i}x_c$, such that:

- $r_{a_i}, r_{b_i}, r_{c_i} \in [0,1]^3$ *and* $r_{a_i} + r_{b_i} + r_{c_i} = 1$,
- $\frac{\partial f^{og}}{\partial x_a}([x],[x],[x]) \geq 0$ *and* $\frac{\partial f^{og}}{\partial x_b}([x],[x],[x]) \leq 0$.

The interval extension by occurrence grouping of f is defined by:

$$[f]_{og}([x]) := [f^{og}]_m([x],[x],[x])$$

Note that f and f^{og} have the same natural evaluation.

In Example 1, we can replace f_1 by f^{og_1} or f^{og_2} in a way respecting the monotonicity constraints of x_a and x_b:

1. $f_1^{og_1}(x_a, x_b) = -(\frac{1}{2}x_a + \frac{1}{2}x_b)3 + 2x_a 2 + 14x_a$
   $\rightarrow \ [f_1^{og_1}]_m([-2,1]) = [-19.875, 16.125]$
2. $f_1^{og_2}(x_a, x_b, x_c) = -x_a^3 + 2(0.25x_a + 0.75x_c)^2 + 14x_a$
   $\rightarrow \ [f_1^{og_2}]_m([-2,1]) = [-20, 16.125]$

Example 2 *Consider the function* $f_2(x) = x^3 - x$ *and the interval* $[x] = [0.5, 2]$. f_2 *is not monotonic and the optimal image* $[f_2]_{opt}([x])$ *is* $[-0.385, 6]$. *The natural evaluation yields* $[-1.975, 7.5]$, *the Horner evaluation* $[-1.5, 6]$. *We can replace* f_2 *by one of the following functions (among others).*

1. $f_2^{og_1}(x_a, x_b) = x_a^3 - (\frac{1}{4}x_a + \frac{3}{4}x_b) \quad \rightarrow \quad [f_2^{og_1}]_m([x]) = [-0.75, 6.375]$
2. $f_2^{og_2}(x_a, x_b) = (\frac{11}{12}x_a + \frac{1}{12}x_b)^3 - x_b \quad \rightarrow \quad [f_2^{og_2}]_m([x]) = [-1.756, 6.09]$

Thus, the new linear program that computes convex linear combinations for achieving occurrence grouping becomes:

Find the values r_{a_i}, r_{b_i} and r_{c_i} for each occurrence x_i that minimize (3) subject to (4), (5), (6) and

$$r_{a_i}, r_{b_i}, r_{c_i} \in [0, 1] \quad \text{for } i = 1, ..., k. \tag{7}$$

Note that this continuous linear program improves the minimum of the objective function because the integrity conditions are relaxed.

Examples

In Example 1, we obtain the minimum 21 and the new function $f_1^{og}(x_a, x_b, x_c) = -x_a^3 + 2(0.25x_a + 0.75x_c)^2 + 14x_a$: $[f_1^{og}]_m([x]) = [-20, 16.25]$. The minimum 21 is inferior to 22 (obtained by the 0,1 linear program). The evaluation by occurrence grouping of f_1 yields $[-20, 16.25]$, which is sharper than the image $[-20, 21]$ obtained by the 0,1 linear program presented in Section 3.

In Example 2, we obtain the minimum 11.25 and the new function $f_2^{og}(x_a, x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b)$. The image $[-0.75, 6.01]$ obtained by occurrence grouping is sharper than the interval computed by natural and Horner evaluations. In this case, the 0,1 linear program of Section 3 yields the naive grouping.

5 An efficient Occurrence Grouping algorithm

Algorithm 1 finds r_{a_i}, r_{b_i}, r_{c_i} that minimize G subject to the constraints. At line 15, the algorithm generates symbolically the new function f^{og} that replaces each occurrence x_i in f by $[r_{a_i}]x_a + [r_{b_i}]x_b + [r_{c_i}]x_c$. Note that the values are represented by thin intervals, of a few u.l.p. large, for taking into account the floating point rounding errors appearing in the computations (see Proposition 4).

Algorithm 1 uses a vector $[g_*]$ of size k containing interval derivatives of f w.r.t. *each occurrence* x_i of x. Each component of $[g_*]$ is denoted by $[g_i]$ and corresponds to the interval $\left[\frac{\partial f}{\partial x_i}\right]([V])$. A symbol indexed by an asterisk refers to a vector (e.g., $[g_*]$, $[r_{a_*}]$).

We illustrate the algorithm using the two univariate functions of our examples: $f_1(x) = -x^3 + 2x^2 + 14x$ and $f_2(x) = x^3 - x$ with domains $[-2, 1]$ and $[0.5, 2]$ respectively.

The interval derivatives of f w.r.t. each occurrence of x have been previously calculated. For these examples, the interval derivatives of f_2 w.r.t. x occurrences are $[g_1] = [0.75, 12]$ and $[g_2] = [-1, -1]$; the interval derivatives of f_1 w.r.t. x occurrences are $[g_1] = [-12, 0]$, $[g_2] = [-8, 4]$ and $[g_3] = [14, 14]$.

Algorithm 1 `Occurrence_Grouping` (**in:** f,$[g_*]$ **out:** f^{og})

1: $[G_0] \leftarrow \sum_{i=1}^{k} [g_i]$
2: $[G_m] \leftarrow \sum_{0 \notin [g_i]} [g_i]$
3: **if** $0 \notin [G_0]$ **then**
4: `OG_case1`$([g_*],[r_{a_*}],[r_{b_*}],[r_{c_*}])$
5: **else if** $0 \in [G_m]$ **then**
6: `OG_case2`$([g_*],[r_{a_*}],[r_{b_*}],[r_{c_*}])$
7: **else**
8: /* $0 \notin [G_m]$ and $0 \in [G_0]$ */
9: **if** $\underline{G_m} \geq 0$ **then**
10: `OG_case3`$^+([g_*],[r_{a_*}],[r_{b_*}],[r_{c_*}])$
11: **else**
12: `OG_case3`$^-([g_*],[r_{a_*}],[r_{b_*}],[r_{c_*}])$
13: **end if**
14: **end if**
15: $f^{og} \leftarrow$ `NewFunction`$(f,[r_{a_*}],[r_{b_*}],[r_{c_*}])$

Algorithm 2 `OG_case2` (**in:** $[g_*]$ **out:** $[r_{a_*}]$, $[r_{b_*}]$, $[r_{c_*}]$)

1: $[G^+] \leftarrow \sum_{[g_i] \geq 0} [g_i]$
2: $[G^-] \leftarrow \sum_{[g_i] \leq 0} [g_i]$
3: $[\alpha_1] \leftarrow \dfrac{\underline{G^+}\overline{G^-} + \overline{G^-}\underline{G^-}}{\underline{G^+}\overline{G^-} - \underline{G^-}\overline{G^+}}$
4: $[\alpha_2] \leftarrow \dfrac{\underline{G^+}\overline{G^+} + \overline{G^-}\underline{G^+}}{\underline{G^+}\overline{G^-} - \underline{G^-}\overline{G^+}}$
5:
6: **for all** $[g_i] \in [g_*]$ **do**
7: **if** $\underline{g_i} \geq 0$ **then**
8: $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (1-[\alpha_1],[\alpha_1],0)$
9: **else if** $\overline{g_i} \leq 0$ **then**
10: $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow ([\alpha_2],1-[\alpha_2],0)$
11: **else**
12: $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (0,0,1)$
13: **end if**
14: **end for**

At line 1, the partial derivative $[G_0]$ of f w.r.t. x is calculated using the sum of the partial derivatives of f w.r.t. each occurrence of x. At line 2, $[G_m]$ gets the value of the partial derivative of f w.r.t. the *monotonic* occurrences of x.

In the examples, for f_1: $[G_0] = [g_1]+[g_2]+[g_3] = [-6,18]$ and $[G_m] = [g_1]+[g_3] = [2,14]$, and for f_2: $[G_0] = [G_m] = [g_1]+[g_2] = [-0.25,11]$.

According to the values of $[G_0]$ and $[G_m]$, we can distinguish 3 cases. The first case is well-known ($0 \notin [G_0]$ in line 3) and occurs when x is a monotonic variable. In the procedure `OG_case1`, *all the occurrences* of x are replaced by x_a (if $[G_0] \geq 0$) or by x_b (if $[G_0] \leq 0$). The evaluation by monotonicity of f^{og} is equivalent to the evaluation by monotonicity of f.

In the second case, when $0 \in [G_m]$ (line 5), the procedure `OG_case2` (Algorithm 2) achieves a grouping of the occurrences of x. Increasing occurrences are replaced by $(1-\alpha_1)x_a + \alpha_1 x_b$, decreasing occurrences by $\alpha_2 x_a + (1-\alpha_2)x_b$ and non monotonic occurrences by x_c (lines 7 to 13 of Algorithm 2).

f_2 falls in this case: $\alpha_1 = \frac{1}{45}$ and $\alpha_2 = \frac{11}{15}$ are calculated at lines 3 and 4 of Algorithm 2 using $[G^+] = [g_1] = [0.75,12]$ and $[G^-] = [g_2] = [-1,-1]$. The new function becomes: $f_2^{og}(x_a,x_b) = (\frac{44}{45}x_a + \frac{1}{45}x_b)^3 - (\frac{11}{15}x_a + \frac{4}{15}x_b)$.

The third case occurs when $0 \notin [G_m]$ and $0 \in [G_0]$. W.l.o.g., assume that $\underline{G_m} \geq 0$. The procedure `OG_case3`$^+$ (Algorithm 3) first groups all the positive and negative occurrences in the increasing group x_a (lines 2–5). The non monotonic occurrences are then replaced by x_a in an order determined by an array *index*[2] (line 7) as long as

[2] An occurrence x_{i_1} is handled before x_{i_2} if $\frac{\overline{g_{i1}} - |[g_{i1}]|}{\underline{g_{i1}}} \geq \frac{\overline{g_{i2}} - |[g_{i2}]|}{\underline{g_{i2}}}$. $index[j]$ yields the index of the j^{th} occurrence in this order.

Algorithm 3 `OG_case3`$^+$(**in:**$[g_*]$ **out:**$[r_{a_*}],[r_{b_*}],[r_{c_*}]$)

1: $[g_a] \leftarrow [0,0]$
2: **for all** $[g_i] \in [g_*]$, $\underline{g_i} \geq 0$ **or** $\overline{g_i} \leq 0$ **do**
3: $\quad [g_a] \leftarrow [g_a] + [g_i]$ /* All positive and negative derivatives are absorbed by $[g_a]$ */
4: $\quad ([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (1,0,0)$
5: **end for**
6:
7: $index \leftarrow$ `descending_sort` $(\{[g_i] \in [g_*], \underline{g_i} < 0\}$, `criterion` $\rightarrow \frac{\overline{g_i} - |[g_i]|}{\underline{g_i}})$
8: $j \leftarrow 1$; $i \leftarrow index[1]$
9: **while** $\underline{g_a} + \underline{g_i} \geq 0$ **do**
10: $\quad ([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (1,0,0)$
11: $\quad [g_a] \leftarrow [g_a] + [g_i]$
12: $\quad j \leftarrow j+1$; $i \leftarrow index[j]$
13: **end while**
14:
15: $[\alpha] \leftarrow -\frac{\underline{g_a}}{\underline{g_i}}$
16: $([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow ([\alpha],0,1-[\alpha])$ /* $[g_a] \leftarrow [g_a] + [\alpha][g_i]$ */
17:
18: $j \leftarrow j+1$; $i \leftarrow index[j]$
19: **while** $j \leq$ `length`$(index)$ **do**
20: $\quad ([r_{a_i}],[r_{b_i}],[r_{c_i}]) \leftarrow (0,0,1)$
21: $\quad j \leftarrow j+1$; $i \leftarrow index[j]$
22: **end while**

the constraint $\sum_{i=1}^{k} r_{a_i}\underline{g_i} \geq 0$ is satisfied (lines 9–13). The criterion varies from 0 (for non monotonic occurrences having $|[g_i]| = \overline{g_i}$) to 1^- (for occurrences having $\overline{g_i} = 0^+$). The first occurrence $x_{i'}$ that cannot be (entirely) replaced by x_a because it would make the constraint (4) unsatisfiable is replaced by $\alpha x_a + (1-\alpha)x_c$, with α such that the constraint is satisfied and equal to 0, i.e., $(\sum_{i=1, i\neq i'}^{k} r_{a_i}\underline{g_i}) + \alpha\underline{g_{i'}} = 0$ (lines 15–16). The rest of the occurrences are replaced by x_c (lines 18–22).

f_1 falls in this case. The increasing and decreasing occurrences of x are first replaced by x_a. The second occurrence of x, that is non monotonic, is then replaced by $\alpha x_a + (1-\alpha)x_c$, where $\alpha = 0.25$ is obtained by forcing the constraint (4) to be 0: $\underline{g_1} + \underline{g_3} + \alpha\underline{g_2} = 0$. The new function is: $f_1^{og}(x_a,x_b,x_c) = -x_a^3 + 2(0.25x_a + 0.75x_c)^2 + 14x_a$.

6 Properties

Proposition 4 *Algorithm 1 (*`Occurrence_grouping`*) is correct and solves the linear program that minimizes (3), modulo floating-point roundings, subject to the constraints (4), (5), (6) and (7).*

We can check that Algorithm 1 respects the four constraints (4)–(7). We have also proven that the minimum of the objective function (3) is reached. The proof concerning `OG_case3` is sophisticated, due to the sort of indices, and uses known results about the

continuous knapsack problem. Special care has been brought to ensure the correctness modulo floating-point roundings. A full proof can be found in the appendix.

Proposition 5 *The time complexity of* `Occurrence_Grouping` *for one variable with k occurrences is* $O(k\,log_2(k))$. *It is time* $O(n\,k\,log_2(k))$ *when a multi-variate function is iteratively transformed by* `Occurrence_Grouping` *for each of its n variables having at most k occurrences each.*

A preliminary gradient calculation by automatic differentiation is time $O(e)$, *where e is the number of unary and binary operators in the expression.*

Computing the gradient of a function amounts in two traversals of the tree representing the mathematical expression [3]. The time complexity of Algorithm 1 is dominated by that of `descending_sort` in the `OG_case3` procedure.

Instead of Algorithm 1, we may use a standard Simplex algorithm providing that the used Simplex implementation takes into account floating-point rounding errors. A performance comparison between Algorithm 1 and Simplex is shown in Section 7.3. Also, as shown in Section 7.2, the time required in practice by `OccurrenceGrouping` is negligible when it is used for solving systems of equations.

Although `Occurrence_Grouping` can be viewed as a heuristic since it minimizes a Taylor-based over-estimate of the function image diameter, it is important to stress that this new interval extension improves the well-known monotonicity-based interval extension.

Proposition 6 *Consider a function* $f : \mathbb{R}^n \to \mathbb{R}$, *and the previously defined interval natural (*$[f]_n$*), monotonicity-based (*$[f]_m$*) and occurrence grouping (*$[f]_{og}$*) extensions of* f. *Let* V *be the n variables involved in* f *with domains* $[V]$. *Then,*

$$[f]_{og}([V]) \subseteq [f]_m([V]) \subseteq [f]_n([V]).$$

7 Experiments

`Occurrence_Grouping` has been implemented in the `Ibex` [5, 4] open source interval-based solver in `C++`. The main goal of these experiments is to show the improvements in CPU time brought by `Occurrence_Grouping` when solving systems of equations.

We briefly recall the combinatorial process followed by an interval-based solver to find all the solutions of a system of equations. The solving process starts from an initial box representing the search space and builds a search tree, following a *Branch & Contract* scheme:

- *Branch:* the current box is *bisected* on one dimension, generating two sub-boxes.
- *Contract:* filtering (also called *contraction*) algorithms reduce the bounds of the box with no loss of solution.

The process terminates with boxes of size smaller than a given positive ω.

Contraction algorithms comprise multidimensional interval Newton algorithms (or variants) issued from the numerical *interval analysis* community [15, 16] along with algorithms from *constraint programming*. In all our experiments, at each node of the search tree, i.e., between two bisections, the solving strategy uses two types of *contractors* (all available in `Ibex`) in sequence:

1. (from constraint programming) the shaving/slicing contractor `3BCID` [12, 17] and a recent *constraint propagation* algorithm, `Mohc` [2, 6], exploiting monotonicity of functions;

2. (from interval analysis) an interval Newton using a Hansen-Sengupta matrix, a left-preconditioning of the matrix, and a Gauss-Seidel method to solve the interval linear system [8].

Sixteen systems of equations have been used in our experiments. They are issued from the COPRIN team website [14], most of them being also known in the COCONUT benchmark suite devoted to interval analysis and global optimization.[3] They correspond to square systems with a finite number of zero-dimensional (isolated) solutions of at least two constraints involving multiple occurrences of variables and requiring more than 1 second to be solved (considering the times appearing in the COPRIN website). All experiments have been performed on a same computer (`Intel 6600 2.4 GHz`).

7.1 Comparison between interval extensions

We first report a comparison between the evaluation by occurrence grouping (i.e., the diameter of $[f]_{og}([V])$) and several existing interval evaluations ($[f]_{ext}$), including Taylor, Hansen and monotonicity-based extensions (see Section 1). Since the Taylor and Hansen extensions are not comparable with the natural extension, to obtain more reasonable comparisons, we have redefined $[f]_t([V]) = [f]'_t([V]) \cap [f]_n([V])$ and $[f]_h([V]) = [f]'_h([V]) \cap [f]_n([V])$, where $[f]'_t$ and $[f]'_h$ are the actual Taylor and Hansen extensions respectively.

Table 1: Different interval extensions compared to $[f]_{og}$

System	$[f]_n$	$[f]_t$	$[f]_h$	$[f]_m$	$[f]_{mr}$	$[f]_{mr+h}$	$[f]_{mr+og}$
`Brent`	0.857	0.985	0.987	0.997	0.998	0.999	1.000
`ButcherA`	0.480	0.742	0.863	0.666	0.786	0.963	1.028
`Caprasse`	0.602	0.883	0.960	0.856	0.953	1.043	1.051
`Direct kin.`	0.437	0.806	0.885	0.875	0.921	0.979	1.017
`Eco9`	0.724	0.785	0.888	0.961	0.980	0.976	1.006
`Fourbar`	0.268	0.718	0.919	0.380	0.427	1.040	1.038
`Geneig`	0.450	0.750	0.847	0.823	0.914	0.971	1.032
`Hayes`	0.432	0.966	0.974	0.993	0.994	0.998	1.001
`I5`	0.775	0.859	0.869	0.925	0.932	0.897	1.005
`Katsura`	0.620	0.853	0.900	0.993	0.999	0.999	1.000
`Kin1`	0.765	0.872	0.880	0.983	0.983	0.995	1.001
`Pramanik`	0.375	0.728	0.837	0.666	0.689	0.929	1.017
`Redeco8`	0.665	0.742	0.881	0.952	0.972	0.997	1.011
`Trigexp2`	0.904	0.904	0.904	0.942	0.945	0.921	1.002
`Trigo1`	0.483	0.766	0.766	0.814	0.814	0.895	1.000
`Virasoro`	0.479	0.738	0.859	0.781	0.795	1.025	1.062
`Yamamura1`	0.272	0.870	0.870	0.758	0.758	0.910	1.000
AVERAGE	0.564	0.822	0.888	0.845	0.874	0.973	1.016

[3] See `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

The list of interval extensions and their related columns in the table are: the natural extension $[f]_n$, the Taylor extension $[f]_t$, the Hansen extension $[f]_h$, the evaluation by monotonicity $[f]_m$, the recursive evaluation by monotonicity $[f]_{mr}$, the recursive evaluation by monotonicity that computes the bounds of the image by using the Hansen extension $[f]_{mr+h}$ and a recursive evaluation by monotonicity that computes the bounds of the image by using the occurrence grouping extension $[f]_{mr+og}$.

The first column of Table 1 indicates the name of each instance. The other columns refer to existing extensions $[f]_{ext}$ and report an average of ratios $\rho_{ext} = \frac{Diam([f]_{og}([V]))}{Diam([f]_{ext}([V]))}$. These ratios are measured while the Branch & Contract solving strategy mentioned above[4] is run to solve the tested system of equations. They are calculated every time a constraint is handled inside the constraint propagation, *at every node of the search tree*, thus avoiding biases.

The table highlights that $[f]_{og}$ computes, in general, sharper interval images than all the competitors. (Only $[f]_{mr+og}$ achieves better evaluations, but it also uses occurrence grouping.) The improvements w.r.t. the two evaluation methods by monotonicity (i.e., $[f]_m$ and $[f]_{mr}$) corroborate the benefits of our approach. For example, in `Fourbar`, $[f]_{og}$ obtains an evaluation diameter which is 42.7% of the evaluation diameter provided by $[f]_{mr}$.[5] $[f]_{mr+h}$ obtains the sharpest evaluations in three benchmarks (`Caprasse`, `Fourbar` and `Virasoro`). However, $[f]_{mr+h}$ is more expensive than $[f]_{og}$. $[f]_{mr+h}$ requires computing $2n$ interval partial derivatives, thus traversing $4n$ times the expression tree if an automatic differentiation method is used [8].

The extension using occurrence grouping $[f]_{mr+og}$ provides necessarily a better evaluation than, or equal to, $[f]_{og}$. However, the experiments on the tested systems show that the gain in evaluation diameter is only 1.6% on average (between 0% and 6.2%), so that we do not believe it constitutes a promising extension due to its additional cost.

7.2 Occurrence Grouping inside a monotonicity-based contractor

These experiments are significant in that they underline the benefits of occurrence grouping for improving the solving of systems of constraints. `Mohc` [2, 6] is a new constraint propagation contractor (like `HC4` or `Box` [3]) that exploits the monotonicity of a function to improve the contraction of the related variable intervals.

Table 2 shows the results of `Mohc` without the `OG` algorithm ($\neg$`OG`), and with `Occurrence_Grouping` (`OG`), i.e., when the function f is transformed into f^{og} before applying the main `MohcRevise(`f^{og}`)` procedure. We observe that, for 7 of the 16 benchmarks, `OccurrenceGrouping` is able to improve significantly the results of `Mohc`; in `Butcher`, `Fourbar`, `Virasoro` and `Yamamura1` the gains in CPU time ($\frac{\neg \texttt{OG}}{\texttt{OG}}$) obtained are greater than 30, 11, 7.5 and 5.4 respectively.

7.3 Practical time complexity

We have first compared the performance of two Occurrence Grouping implementations: using our ad-hoc algorithm (`Occurrence_Grouping`) and using a Simplex method. The basic Simplex implementation [7] we have used is not rigorous, i.e., it does not take into account rounding errors due to floating point arithmetic. Adding

[4] Similar results are obtained by other strategies.

[5] Most of the results are close to 1, which seems to show that there exist no significant differences between the different evaluations. However, it is known that even if a few boxes are sharply evaluated, this can then avoid a lot of bisections in the solving process and improve significantly the performance of the solver.

Table 2: Experimental results using the monotonicity-based contractor `Mohc` inside a solving strategy. The first column indicates the name of each instance, along with its number of variables/equations (left) and solutions (right). The second column reports the CPU time (first row of a multi-row) and the number of nodes (second row) obtained by a strategy using `3BCID(Mohc)` without `OG`. The third column report the results of our strategy using `3BCID(OG+Mohc)`. The fourth column indicates the (large) number of calls to `Occurrence_Grouping` during the solving process.

System	`Mohc`		
	¬`OG`	`OG`	#`OG` calls
`ButcherA` 8 3	>1 day	**1722** 288,773	16,772,045
`Brent` 10 1008	20 3811	20.3 3805	30,867
`Caprasse` 4 18	2.57 1251	2.71 867	60,073
`Eco9` 9 16	13.31 6161	13.96 6025	70,499
`Fourbar` 4 3	4277 1,069,963	**385** 57,377	8,265,730
`Geneig` 6 10	328 76,465	**111** 13,705	2,982,275
`Hayes` 8 1	17.62 4599	17.45 4415	5316
`I5` 10 30	57.25 10,399	58.12 9757	835,130
`Katsura` 12 7	100 3711	103 3625	39,659
`Kin1` 6 16	1.82 85	1.79 83	316
`Pramanik` 3 2	67.98 51,877	**21.23** 12,651	395,083
`Redeco8` 8 8	5.98 2285	6.12 2209	56,312
`Trigexp2` 11 0	90.5 14,299	88.2 14301	338,489
`Trigo1` 10 9	137 1513	**57** 443	75,237
`Virasoro` 8 24	6790 619,471	**901** 38,389	5,633,140
`Yamamura1` 8 7	11.59 2663	**2.15** 343	43,589

this feature should make the algorithm run more slowly. Two important results have been obtained. First, we have checked experimentally that our algorithm is correct, i.e., it obtains the minimum value for the objective function G. Second, just as we expected, the performance of the general-purpose Simplex method is worse than the performance of our algorithm. It runs between 2.32 (`Brent`) and 10 (`Virasoro`) times more slowly.

Two results highlight that the CPU time required by `Occurrence_Grouping` is very interesting in practice. In Table 2 indeed, for instances like `Brent`, `Eco9` or `Trigexp2`, `Occurrence_Grouping` is called a large number of times with roughly no effect on solving: similar number of choice points is reported with and without occurrence grouping. However, the overall CPU is also very similar. The same observation has been made in another experiment reported in Section 5.6.1 of Araya's PhD thesis [1].

8 Conclusion

We have proposed a new method to improve the monotonicity-based evaluation of a function f. This *Occurrence Grouping* method creates for each variable of f three auxiliary, respectively increasing, decreasing and non monotonic variables. It then transforms f into a function f^{og} that replaces the occurrences of every variable by a convex linear combination of these auxiliary variables. The evaluation by monotonicity of f^{og} defines the *evaluation by occurrence grouping* of f and is better than the evaluation by monotonicity of f.

The extension by occurrence grouping computes at a low cost sharper interval images than existing interval extensions do. The main benefits of occurrence grouping lie in the improvement of an efficient contraction algorithm, called `Mohc`, that exploits monotonicity of functions. Occurrence grouping transforms, with nearly no overhead in practice, the constraints on the fly, during the constraint propagation and the solving process.

References

[1] I. Araya. *Exploiting Common Subexpressions and Monotonicity of Functions for Filtering Algorithms over Intervals*. PhD thesis, University of Nice–Sophia, 2010.

[2] I. Araya, G. Trombettoni, and B. Neveu. Exploiting Monotonicity in Interval Constraint Propagation. In *Proc. AAAI*, pages 9–14. AAAI Press, 2010.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.

[4] G. Chabert. Ibex – An Interval Based EXplorer. `www.ibex-lib.org`, 2010.

[5] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.

[6] G. Chabert and L. Jaulin. Hull Consistency Under Monotonicity. In *Proc. Constraint Programming CP, LNCS 5732*, pages 188–195, 2009.

[7] B.P. Flannery, W. Press, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C*. Press Syndicate of the University of Cambridge, New York, 1992.

[8] E. Hansen. *Global Optimization using Interval Analysis*. Marcel Dekker inc., 1992.

[9] William G. Horner. A new Method of Solving Numerical Equations of all Orders, by Continuous Approximation. *Philosophical Transactions of the Royal Society of London*, 109:308–335, 1819.

[10] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[11] V. Kreinovich, A.V. Lakeyev, J. Rohn, and P.T. Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations*. Kluwer, 1997.

[12] O. Lhomme. Consistency Techniques for Numeric CSPs. In *Proc. IJCAI*, pages 232–238, 1993.

[13] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[14] J.-P. Merlet. The COPRIN examples page. `www-sop.inria.fr/coprin/logiciels/ALIAS/ Benches/benches.html`, 2010.

[15] R. Moore. *Interval Analysis*. Prentice Hall, 1966.

[16] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[17] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.

A Proof of Proposition 4 (1 of 2): Algorithm 1 is correct

Proof

Due to Lemmas 1 and 2, there exist real values $r^*_{a_i}$, $r^*_{b_i}$ and $r^*_{c_i}$ inside the intervals computed by Algorithm 1, such that the monotonicity conditions w.r.t. variables x_a and x_b are satisfied.

Let us call $f^{og*}(x_a,x_b,x_c)$ the function $f(x)$ in which every occurrence x_i is replaced by the convex linear combination $r^*_{a_i}x_a + r^*_{b_i}x_b + r^*_{c_i}x_c$. The evaluation by monotonicity of f^{og*} over intervals $[x_a] = [x_b] = [x_c] = [x]$ is thus an overestimate of the hull of the image by f of all $x \in [x]$. The evaluation by monotonicity of the actual function f^{og} computed by Algorithm 1 using intervals $[r_{a_i}]$, $[r_{b_i}]$ and $[r_{c_i}]$ is an overestimate of the evaluation of f^{og*} and thus is also an overestimate of the hull of the image by f of all $x \in [x]$. □

Lemma 1 *Algorithm 2 (*`OG_case2`*) computes, for every occurrence i, intervals* $[r_{a_i}]$, $[r_{b_i}]$ *and* $[r_{c_i}]$ *that satisfy the constraints:*

$$(\forall i = 1\ldots k)(\exists r_{a_i} \in [r_{a_i}])(\exists r_{b_i} \in [r_{b_i}])(\exists r_{c_i} \in [r_{c_i}]) s.t. \quad \begin{array}{l} r_{a_i}, r_{b_i}, r_{c_i} \in [0,1] \\ r_{a_i} + r_{b_i} + r_{c_i} = 1 \\ \sum_{i=1}^{k} \underline{g_i} r_{a_i} \geq 0 \\ \sum_{i=1}^{k} \overline{g_i} r_{b_i} \leq 0 \end{array}$$

Proof

For every occurrence i, according to conditions met by $[g_i]$, for r_{a_i} we choose the value $1-\alpha_1$ (line 8), α_2 (line 10) or 0 (line 12); for r_{b_i}, we choose α_1, $1-\alpha_2$ or 0; for r_{c_i}, we choose 0 or 1 (that are both reals and floats); where α_1 and α_2 are the following two real numbers (not necessarily floating point numbers):

$$\alpha_1 = \frac{\underline{G^+}\overline{G^-} + \overline{G^-}\underline{G^-}}{\underline{G^+}\overline{G^-} - \underline{G^-}\overline{G^+}} \quad \textit{and} \quad \alpha_2 = \frac{\underline{G^+}\overline{G^+} + \overline{G^-}\underline{G^+}}{\underline{G^+}\overline{G^-} - \underline{G^-}\overline{G^+}}$$

First, thanks to the conservative interval-based computation of $[\alpha_1]$ and $[\alpha_2]$ performed at lines 3 and 4, we have $\alpha_1 \in [\alpha_1]$ and $\alpha_2 \in [\alpha_2]$.

Second, checking that $r_{a_i}, r_{b_i}, r_{c_i} \in [0,1]$ requires $0 \leq \alpha_1 \leq 1$ and $0 \leq \alpha_2 \leq 1$. This can be proved using some inequality properties of the parameters: $\overline{G^+} \geq \underline{G^+}$ and $\overline{G^-} \geq \underline{G^-}$ (inherent property of intervals); $\underline{G^+} \geq 0$, and $\overline{G^-} \leq 0$; $-\overline{G^-} \leq \overline{G^+}$ and $\underline{G^+} \leq -\underline{G^-}$ (deduced by the condition $0 \in [G_m]$ where $[G_m] = [G^+] + [G^-]$).

Third, this is straightforward to check at lines 8, 10 and 12 that the chosen values for r_{a_i}, r_{b_i}, r_{c_i} verify the relation $r_{a_i} + r_{b_i} + r_{c_i} = 1$.

Finally, the main difficulty consists in proving that, for any occurrence i:

$$\sum_{i=1}^{k} \underline{g_i} r_{a_i} \geq 0 \quad \textit{and} \quad \sum_{i=1}^{k} \overline{g_i} r_{b_i} \leq 0.$$

First, α_1 and α_2 are computed by analytically solving the equations:

$$\begin{array}{rcl} (1-\alpha_1) \times \underline{G^+} + \alpha_2 \times \underline{G^-} & = & 0 \\ \alpha_1 \times \overline{G^+} + (1-\alpha_2) \times \overline{G^-} & = & 0 \end{array}$$

The values $\underline{G^+}$, $\overline{G^+}$, $\underline{G^-}$ and $\overline{G^-}$ are computed in lines 1 and 2 of the algorithm. Due to floating point errors, these values are overestimated, i.e.:

$$\underline{G^+} \leq \sum_{[g_i]\geq 0} \underline{g_i}, \quad \overline{G^+} \geq \sum_{[g_i]\geq 0} \overline{g_i}, \quad \underline{G^-} \leq \sum_{[g_i]\leq 0} \underline{g_i}, \quad \overline{G^-} \geq \sum_{[g_i]\leq 0} \overline{g_i}.$$

Thus, we obtain:

$$\begin{aligned}
\sum_{i=1}^{k} \underline{g_i} r_{a_i} &= (1-\alpha_1) \sum_{[g_i]\geq 0} \underline{g_i} + \alpha_2 \sum_{[g_i]\leq 0} \underline{g_i} \quad \geq \quad (1-\alpha_1) \times \underline{G^+} + \alpha_2 \times \underline{G^-} = 0 \\
\sum_{i=1}^{k} \overline{g_i} r_{b_i} &= \alpha_1 \sum_{[g_i]\geq 0} \overline{g_i} + (1-\alpha_2) \sum_{[g_i]\leq 0} \overline{g_i} \quad \leq \quad \alpha_1 \times \overline{G^+} + (1-\alpha_2) \times \overline{G^-} = 0
\end{aligned}$$

□

Lemma 2 *Algorithm 3 (*`OG_case3+`*), for every occurrence i, computes intervals* $[r_{a_i}]$, $[r_{b_i}]$ *and* $[r_{c_i}]$ *that satisfy the constraints:*

$$(\forall i = 1 \ldots k)(\exists r_{a_i} \in [r_{a_i}])(\exists r_{b_i} \in [r_{b_i}])(\exists r_{c_i} \in [r_{c_i}]) s.t. \quad \begin{array}{r} r_{a_i}, r_{b_i}, r_{c_i} \in [0,1] \\ r_{a_i} + r_{b_i} + r_{c_i} = 1 \\ \sum_{i=1}^{k} \overline{g_i} r_{b_i} \leq 0 \\ \sum_{i=1}^{k} \underline{g_i} r_{a_i} \geq 0 \end{array}$$

Proof

Let us denote by $i' = index(j')$ the occurrence i studied at lines 15, 16 of the algorithm. We distinguish three main cases: $i < i'$ (lines 2–13), $i > i'$ (lines 18–21) and $i = i'$ (lines 15–16):

- $i < i'$: We choose $(r_{a_i}, r_{b_i}, r_{c_i}) = (1,0,0)$.
  (Recall that 0 and 1 are floating point numbers.)
- $i > i'$ ($i \in \{index(j'+1), \ldots, index(k)\}$):
  We choose $(r_{a_i}, r_{b_i}, r_{c_i}) = (0,0,1)$.
- $i = i'$: We choose $(r_{a'_i}, r_{b'_i}, r_{c'_i}) = (\underline{\alpha}, 0, 1-\underline{\alpha})$. That is, for $r_{a'_i}$, we select the lower bound $\underline{\alpha}$ of the interval $[\alpha]$ computed in line 15 of the algorithm.

First, this is straightforward to check in the first two cases that $r_{a_i}, r_{b_i}, r_{c_i} \in [0,1]$. When $i = i'$, at lines 15, 16 of the algorithm, we need to prove that $\alpha \in [0,1]$. However, we have $\underline{g_a} \geq 0$, $\underline{g_i} \leq 0$ (because $0 \in [g_i]$) and $\underline{g_a} + \underline{g_i} < 0$ (because the condition of the `while` loop just failed). Therefore, $\alpha = -\frac{\underline{g_a}}{\underline{g_i}} \in [0,1]$.

Second, this is straightforward to check in the three cases that the chosen values for r_{a_i}, r_{b_i}, r_{c_i} verify the relation $r_{a_i} + r_{b_i} + r_{c_i} = 1$.

Third, this is also straightforward to check $\sum_{i=1}^{k} \overline{g_i} r_{b_i} \leq 0$ since, for all i, we always select the value 0 for r_{b_i}.

Finally, the main difficulty consists in proving $\sum_{i=1}^{k} \underline{g_i} r_{a_i} \geq 0$, for any occurrence i.

We have: $\sum_{i=1}^{k} \underline{g_i} r_{a_i} = \sum_{[g_i]\geq 0} \underline{g_i} + \sum_{\substack{i=index(j) \\ j=1}}^{j=j'-1} \underline{g_i} + \underline{g_{i'}} r_{a_{i'}}$.

Let $\underline{\dot{g_a}}$ denote the real number $\sum_{[g_i]\geq 0} \underline{g_i} + \sum_{\substack{i=index(j) \\ j=1}}^{j=j'-1} \underline{g_i}$ ($\underline{\dot{g_a}}$ is not necessarily a floating point number.) By construction, we have $\underline{\dot{g_a}} + \underline{g_{i'}}\alpha = 0$, where $\alpha = -\frac{\underline{\dot{g_a}}}{\underline{g_{i'}}}$. Indeed, $\underline{\dot{g_a}} - \underline{g_{i'}}\frac{\underline{\dot{g_a}}}{\underline{g_{i'}}} = 0$.

However, the real number α does not necessarily belong to the interval $[\alpha] = \frac{\underline{g_a}}{\underline{g_{i'}}}$ computed at line 15 (because of the overestimate induced by rounding errors in the sum $\sum_{[g_i]\geq 0} \underline{g_i} + \sum_{\substack{i=index(j) \\ j=1}}^{j=j'-1} \underline{g_i}$ over the floats).

Fortunately, we have $0 \leq \underline{g_a} \leq \underline{\dot{g_a}} = \sum_{[g_i]\geq 0} \underline{g_i} + \sum_{\substack{i=index(j) \\ j=1}}^{j=j'-1} \underline{g_i}$ and thus $\underline{\alpha} \leq \alpha$.

Finally, recalling that $g_{i'} \leq 0$, we obtain:

$$\sum_{i=1}^{k} \underline{g_i} r_{a_i} = \underline{\dot{g_a}} + \underline{g_{i'}}\underline{\alpha} \geq \underline{\dot{g_a}} + \underline{g_{i'}}\alpha = 0$$

□

B Proof of Proposition 4 (2 of 2): Algorithm 1 is optimal

Proof

The proof is subdivided into two parts:

Algorithm 2 solves the linear program that minimizes (3) if $0 \in [G_m]$.
Following Lemma 3, we can rewrite the objective function as $G = \texttt{Diam}([G_0]) + G_1 + G_2$, where $G_1 = \underline{g_a} - \overline{g_b}$ and $G_2 = \sum_{i=1}^{k} \left((|[g_i]| - \texttt{Diam}([g_i]) r_{c_i}) \right)$. The term $\texttt{Diam}([G_0])$ is constant; Lemmas 4 and 5 show that the algorithm finds values r_{a_i}, r_{b_i} and r_{c_i} that minimize G_1 and G_2 resp. Thus, Algorithm 2 finds an optimal solution for G.

Algorithm 3 solves the linear program that minimizes (3) if $0 \notin [G_m]$ *and* $0 \in [G_0]$.
According to Lemma 6, the minimum of G can be found only if $\forall i: \quad r_{b_i} = 0$. Then, following Lemma 7, the problem can be transformed into the fractional knapsack problem instance:
$FKP(M, \{w_1, ..., w_k\}, \{v_1, ..., v_k\})$, where $v_i = -(\overline{g_i} - |[g_i]|)$, $w_i = -\underline{g_i}$ and $M = \sum_{i=1}^{k_1} \underline{g_i}$.
Proposition B shows that a greedy algorithm that puts the items sorted by $\frac{v_i}{w_i}$ is able to reach the optimal solution. Algorithm 3 does exactly that.

□

Lemma 3 *The objective function (3) can be rewritten as:*

$$G = \texttt{Diam}([G_0]) + \underline{g_a} - \overline{g_b} - \texttt{Diam}([g_c]) + \sum_{i=1}^{k} \left(|[g_i]| r_{c_i} \right) \tag{8}$$

where $[G_0] = \sum_{i=1}^{k} [g_i]$ *and* $[g_c] = \sum_{i=1}^{k} \left([g_i] r_{c_i} \right)$

Proof

Observe that $[G_0] = [g_a] + [g_b] + [g_c]$ and using interval arithmetic properties $\texttt{Diam}([G_0]) = \texttt{Diam}([g_a]) + \texttt{Diam}([g_b]) + \texttt{Diam}([g_c])$.

Taking into account the constraints (4) and (5): $\texttt{Diam}([g_a]) = \overline{g_a} - \underline{g_a}$ and $\texttt{Diam}([g_b]) = -\underline{g_b} + \overline{g_b}$. The diameter of $[G_0]$ can be written: $\texttt{Diam}([G_0]) = \overline{g_a} - \underline{g_b} - (\underline{g_a} - \overline{g_b}) + \texttt{Diam}([g_c])$. Replacing $\overline{g_a} - \underline{g_b}$ in (3) by $\texttt{Diam}([G_0]) + \underline{g_a} - \overline{g_b} + \texttt{Diam}([g_c])$ we obtain (8). □

Lemma 4 *Let P_1 be the following linear program: find values r_{a_i}, r_{b_i}, and r_{c_i} for all $i = 1, ..., k$ that minimize $G_1 = \underline{g_a} - \overline{g_b}$ subject to (4), (5),(6) and (7) (see page 8). If $0 \in [G_m]$, then Algorithm 2 finds an optimal solution to P_1.*

Proof

Due to constraints (4) and (5), we deduce that $G_1 \geq 0$.

If we analyze the `for` loop (lines 6 to 14) of Algorithm 2 we note that each variable r_{a_i} gets the value $1 - \alpha_1$ if $[g_i] \geq 0$, gets the value α_2 if $[g_i] < 0$; r_{a_i} gets the value 0 otherwise. Then, at the end of the `for` loop we can compute:

$$\underline{g_a} = (1 - \alpha_1) \sum_{[g_i] \geq 0} \underline{g_i} + \alpha_2 \sum_{[g_i] < 0} \underline{g_i} \tag{9}$$

As $\underline{G^+} = \sum_{[g_i] \geq 0} \underline{g_i}$ and $\underline{G^-} = \sum_{[g_i] < 0} \underline{g_i}$ (lines 1-2 of the algorithm), Equation 9 is reduced to: $\underline{g_a} = (1 - \alpha_1)\underline{G^+} + \alpha_2 \underline{G^-}$. Then replacing α_1 and α_2 by their values in lines 3-4 of the algorithm we obtain $\underline{g_a} = 0$ (in the same way we obtain $\overline{g_b} = 0$) respecting constraints (4) and (5) and finding the minimum of $G_1 = 0$. Constraint (6) remains satisfiable in lines 8, 10 and 12 of the algorithm.

□

Lemma 5 *Let P_2 be the linear program: find values r_{a_i}, r_{b_i}, and r_{c_i} for all $i = 1, ..., k$ that minimize $G_2 = \sum_{i=1}^{k} \left((|[g_i]| - \texttt{Diam}([g_i])) r_{c_i} \right)$, subject to (6) and (7). Algorithm 2 finds an optimal solution to P_2.*

Proof

$|[g_i]| - \texttt{Diam}([g_i])$ is less than 0 iff $0 \in [g_i]$. Then, to minimize G_2, it is enough that, for each $[g_i]$ containing 0, $r_{c_i} = 1$. For respecting constraints (6) and (7), we add $r_{a_i} = 0$ and $r_{b_i} = 0$. This setting is achieved at line 12 of Algorithm 2. □

Definition 4 (Fractional knapsack problem) *Given a knapsack with capacity M and a list of k elements with weights $W = \{w_1, ..., w_k\}$ and values $V = \{v_1, ..., v_k\}$. The*

fractional knapsack problem $FKP(M,W,V)$ *consists in finding the set* $R = \{r_1,...,r_n\}$ *that maximizes the total value* $\sum_{i=1}^{k} v_i r_i$ *subject to* $M - \sum_{i=1}^{k} w_i r_i \geq 0$ *and* $r_i \in [0,1]$.

Proof

[13]
Let $FKP(M,W,V)$ an instance of the fractional knapsack problem. W.l.o.g., assume that W and P are sorted in decreasing order of the ratio $\frac{v_i}{w_i}$. It means that the given sets are sorted such that $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq ... \geq \frac{v_k}{w_k}$. The optimal solution is found when:

$$\begin{aligned} r_i &= 1 \quad \text{for } i = 1,...,s-1 \\ r_s &= \frac{M - \sum_{i=1}^{s-1} w_i}{w_s} \\ r_j &= 0 \quad \text{for } j = s+1,...,k \end{aligned}$$

□ In other words, elements are added to the solution set in this decreasing order until the capacity M is reached or all the elements are used. Observe that there is at most one fractionary element in the solution set (r_s). All others either are full elements ($r_i = 1$) or do not belong to the solution ($r_j = 0$).

Lemma 6 *Let G be the objective function:*

$$G = \overline{g_a} - \underline{g_b} + \sum_{i=1}^{k} (|[g_i]|r_{c_i}) \tag{10}$$

subject to the constraints (4), (5),(6) and (7) (see page 8). If $\underline{G_m} \geq 0$ *(where* $[G_m]$ *is the sum of the partial derivatives of the monotonic occurrences, i.e.,* $[G_m] = \sum_{0 \notin [g_i]} [g_i]$*), then for all grouping* og_1 *with* $\sum_{i=0}^{k} r_{b_i} \neq 0$ *(i.e., the set of occurrences* x_b *is not empty) there exists a grouping* og_2*, such that* $G(og_2) \leq G(og_1)$.

Proof

Consider a grouping og_1 such that the partial derivatives w.r.t. x_a, x_b and x_c are given resp. by:

$$\begin{aligned} [g_a^1] &= [g_a^m] + [g_a^0] \\ [g_b^1] &= [g_b^m] + [g_b^0] \\ [g_c^1] &= [g_c^m] + [g_c^0] \end{aligned}$$

where $[g_a^m]$ (resp. $[g_b^m]$ and $[g_c^m]$) corresponds to the sum of the partial derivatives of the monotonic occurrences of x_a (resp. x_b and x_c), i.e., $[g_a^m] = \sum_{0 \notin [g_i]} ([g_i] r_{a_i})$. $[g_a^0]$ (resp. $[g_b^0]$ and $[g_c^0]$) corresponds to the sum of the partial derivatives of the nonmonotonic

occurrences of x_a (resp. x_b and x_c), i.e., $[g_a^0] = \sum_{0\in[g_i]} ([g_i]r_{a_i})$. We can compute $G(og_1)$ using (10):

$$G(og_1) = \overline{g_a^m} + \overline{g_a^0} - (\underline{g_b^m} + \underline{g_b^0}) + \sum_{0\notin[g_i]} (|[g_i]|r_{c_i}) + \gamma$$

where

$$\gamma = \sum_{0\in[g_i]} (|[g_i]|r_{c_i})$$

Consider a grouping og_2 such that the partial derivatives w.r.t. x_a, x_b and x_c are given resp. by:

$$\begin{array}{rcl} [g_a^2] & = & [g_a^m] + [g_b^m] + [g_c^m] + \alpha \times [g_a^0] \\ [g_b^2] & = & [0,0] \\ [g_c^2] & = & [g_c^0] + (1-\alpha) \times [g_a^0] + [g_b^0] \end{array}$$

where $0 \leq \alpha \leq 1$. Creating the grouping og_2 from og_1 is always possible. A way consists in "moving" all the monotonic occurrences, from x_b and x_c, to x_a. Maybe, we would also need to move a fraction $(1-\alpha)$ of each non monotonic occurrence from x_a to x_c for keeping satisfied the constraint (4), i.e. $\underline{g_a^1} \geq 0$.

We compute $G(og_2)$:

$$G(og_2) = \overline{g_a^m} + \overline{g_b^m} + \overline{g_c^m} + \alpha \times \overline{g_a^0} + \gamma + \beta$$

where

$$\beta = (1-\alpha) \sum_{0\in[g_i]} (|[g_i]|r_{a_i}) + \sum_{0\in[g_i]} \left(|[g_i]|r_{b_i}\right)$$

As $0 \in [g_i]$ implies $|[g_i]| \leq (\overline{g_i} - \underline{g_i})$:

$$\beta \leq (1-\alpha) \sum_{0\in[g_i]} \left((\overline{g_i} - \underline{g_i})r_{a_i}\right) + \sum_{0\in[g_i]} \left((\overline{g_i} - \underline{g_i})r_{b_i}\right))$$

Finally,

$$\beta \leq (1-\alpha)(\overline{g_a^0} - \underline{g_a^0}) + (\overline{g_b^0} - \underline{g_b^0})$$

Now we can compute the subtraction $G(og_1) - G(og_2)$:

$$\begin{array}{rcl} G(og_1) - G(og_2) & = & \overline{g_a^m} + \overline{g_a^0} - \underline{g_b^m} - \underline{g_b^0} + \sum_{0\notin[g_i]} (|[g_i]|r_{c_i}) + \gamma - \overline{g_a^m} - \overline{g_b^m} - \overline{g_c^m} - \alpha \times \overline{g_a^0} - \gamma - \beta \\ G(og_1) - G(og_2) & = & \overline{g_a^0} - \underline{g_b^m} - \underline{g_b^0} + \sum_{0\notin[g_i]} (|[g_i]|r_{c_i}) - \overline{g_b^m} - \overline{g_c^m} - \alpha \times \overline{g_a^0} - \beta \end{array}$$

If we use the upper bound of β:

$$\begin{array}{rcl} G(og_1) - G(og_2) & \geq & \overline{g_a^0} - \underline{g_b^m} - \underline{g_b^0} + \sum_{0\notin[g_i]} (|[g_i]|r_{c_i}) - \overline{g_b^m} - \overline{g_c^m} - \alpha \times \overline{g_a^0} - (1-\alpha)(\overline{g_a^0} - \underline{g_a^0}) - \overline{g_b^0} + \underline{g_b^0} \\ G(og_1) - G(og_2) & \geq & -\underline{g_b^m} + \sum_{0\notin[g_i]} (|[g_i]|r_{c_i}) - \overline{g_b^m} - \overline{g_c^m} + (1-\alpha)\underline{g_a^0} - \overline{g_b^0} \end{array}$$

We can write the constraints (4), (5) for the grouping og_1:

$$\underline{g_a^m} + \underline{g_a^0} \geq 0 \quad (11)$$

$$-\overline{g_b^m} - \overline{g_b^0} \geq 0 \quad (12)$$

It is trivial that if $0 \notin [g_i]$, then $|[g_i]| - (\overline{g_i} - \underline{g_i}) \geq 0$. Extending this relation we can obtain:

$$\sum_{0 \notin [g_i]} (|[g_i]|r_{c_i}) - (\overline{g_c^m} - \underline{g_c^m}) \geq 0 \tag{13}$$

It is also trivial to verify that if $0 \notin [g_i]$, then $|[g_i]| - \overline{g_i} \geq 0$. Extending this relation we can obtain:

$$\sum_{0 \notin [g_i]} (|[g_i]|r_{c_i}) - \overline{g_c^m} \geq 0 \tag{14}$$

According to the constraint (4) in the grouping og_2, the value of α must satisfy:

$$\alpha \leq \frac{-(\underline{g_a^m} + \underline{g_b^m} + \underline{g_c^m})}{\underline{g_a^0}}$$

The right side of the constraint is always positive: $\underline{g_a^m} + \underline{g_b^m} + \underline{g_c^m} = \underline{G_m} > 0$ and $\underline{g_a^0} < 0$. Thus, we can identify two cases:

- When the right side is inferior to one, if we set α to $\frac{-(\underline{g_a^m} + \underline{g_b^m} + \underline{g_c^m})}{\underline{g_a^0}}$ (i.e., $(1-\alpha)\underline{g_a^0} = \underline{g_a^0} + \underline{g_a^m} + \underline{g_b^m} + \underline{g_c^m}$), then we obtain in (11):

$$\begin{aligned} G(og_1) - G(og_2) &\geq -\underline{g_b^m} + \sum_{0 \notin [g_i]} (|[g_i]|r_{c_i}) - \overline{g_b^m} - \overline{g_c^m} + \underline{g_a^0} + \underline{g_a^m} + \underline{g_b^m} + \underline{g_c^m} - \overline{g_b^0} \\ G(og_1) - G(og_2) &\geq \left(\sum_{0 \notin [g_i]} (|[g_i]|r_{c_i}) - (\overline{g_c^m} - \underline{g_c^m}) \right) + (\underline{g_a^m} + \underline{g_a^0}) + (-\overline{g_b^m} - \overline{g_b^0}) \end{aligned}$$

  Observe that the three expressions into parentheses are positive according to (11), (12) and (13). Then $G(og_1) \geq G(og_2)$.

- When the right side is greater than one, if we set α to 1, then we obtain in (11):

$$\begin{aligned} G(og_1) - G(og_2) &\geq -\underline{g_b^m} + \sum_{0 \notin [g_i]} (|[g_i]|r_{c_i}) - \overline{g_b^m} - \overline{g_c^m} - \overline{g_b^0} \\ G(og_1) - G(og_2) &\geq -\underline{g_b^m} + (-\overline{g_b^m} - \overline{g_b^0}) + (\sum_{0 \notin [g_i]} (|[g_i]|r_{c_i}) - \overline{g_c^m}) \end{aligned}$$

  Observe that the first term in the left side $(-\underline{g_b^m})$ must be positive to satisfy the constraint (12), the second and third terms are positive according to (12) and (14). Thus, $G(og_1) \geq G(og_2)$. □

Lemma 7 *Let P_3 be the following linear program: find values r_{a_i} r_{b_i} and r_{c_i} for all $i = 1..k$ that minimize $G_3 = \texttt{Diam}([G_0]) + \underline{g_a} - \overline{g_b} - \texttt{Diam}([g_c]) + \sum_{i=1}^{k} |[g_i]|r_{c_i}$ subject to (4), (5),(6) and (7) (see page 8). If $\forall i: \; r_{b_i} = 0$, then P_3 is equivalent to the following fractional knapsack problem:*

(W.l.o.g., assume that the occurrences are sorted: only the first k_1 occurrences have a positive partial derivative.) Find the values r_{a_i}, r_{c_i} for $i = k_1 + 1, ..., k$ that maximize

$G'_3 = \sum_{i=k_1+1}^{k} v_i r_{a_i}$, *subject to:*

$$\begin{aligned} M - \sum_{i=k_1+1}^{k} w_i r_{a_i} &\geq 0 \\ r_{a_i} &\in [0,1] \\ r_{c_i} &= 1 - r_{a_i} \end{aligned}$$

where $v_i = -(\overline{g_i} - |[g_i]|)$, $w_i = -\underline{g_i}$ *and* $M = \sum_{i=1}^{k_1} \underline{g_i}$.

Proof

The condition $\forall i: \quad r_{b_i} = 0$ implies that $\overline{g_b} = 0$. Thus, we can rewrite the objective function:

$$G_3 = \texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(\underline{g_i} r_{a_i} + \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right) r_{c_i}\right)$$

As $r_{a_i} + r_{c_i} = 1$, the expression is equivalent to:

$$G_3 = \texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(\underline{g_i} r_{a_i} + \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right)(1 - r_{a_i})\right)$$

$$G_3 = \texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right) + \sum_{i=1}^{k} \left(\underline{g_i} - \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right) r_{a_i}\right)$$

After simplification, we finally obtain:

$$G_3 = \texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right) + \sum_{i=1}^{k} \left((\overline{g_i} - |[g_i]|) r_{a_i}\right)$$

Observe that if $[g_i] \geq 0$, $|[g_i]| = \overline{g_i}$, thus, $G_3 = \texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right)$. In other words, if $[g_i] \geq 0$, the objective function does not depend on the values of r_{a_i} and r_{c_i}. Thus, we set $r_{a_i} = 1$, for all i with $[g_i] \geq 0$, because we maximize this way the relaxation of the constraint (4):

$$\sum_{i=1}^{k} \underline{g_i} r_{a_i} \geq 0$$

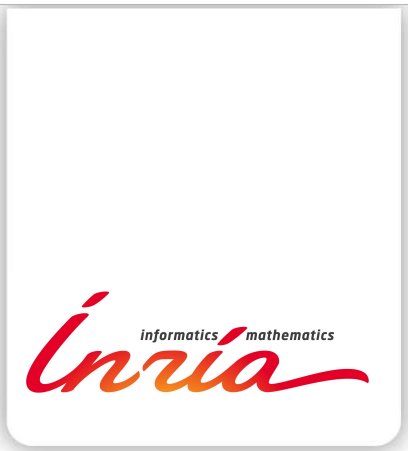
As only the first k_1 occurrences have a positive partial derivative, we can rewrite the objective function as follows:

$$G_3 = \texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right) + \sum_{i=k_1+1}^{k} (\overline{g_i} - |[g_i]|) r_{a_i}$$

and the constraint (4):

$$\sum_{i=1}^{k_1} \underline{g_i} - \sum_{i=k_1+1}^{k} (-\underline{g_i}) r_{a_i} \geq 0$$

Note that $\texttt{Diam}([G_0]) + \sum_{i=1}^{k} \left(-(\overline{g_i} - \underline{g_i}) + |[g_i]|\right)$ and $\sum_{i=1}^{k_1} \underline{g_i}$ are constants, thus the problem can be rewritten as the fractional knapsack problem of Lemma 7. □



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399